



Democratizing AI in Software Development

From Few-Shot Testing to Trustworthy Benchmarks and Accessible AI Tools

Jeremy S. Bradbury

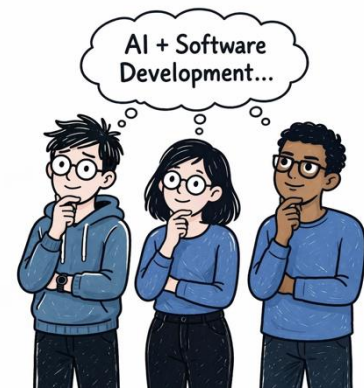
Software Engineering & Education Research Lab
Ontario Tech University, Oshawa, Canada

<http://www.seerlab.ca>

SEER*LAB

| mairi

| OntarioTech
UNIVERSITY



Collaborators



Daniel Hinbest, Bridget Green, Riddhi More, Samantha Navarro, Stacey Koornneef, Rosie Khurmi, Michael Miljanovic, Nadia Goralski, André Wemans , Adam Kolodziejczak, Mosarrat Rumman, Bisha Fatima.

What Limits Democratization of AI in Software Development?



**Lack of Resources
(data, computing)**



**Lack of Trust in
Results**



**Lack of Expertise
in Technology**

Lack of Resources

- **Data** → Lack of data need to train or fine-tune the model
- **Computing** → Lack of computing resources for model creation and fine-tuning



An Analysis of LLM Fine tuning and Few- Shot Learning for Flaky Test Detection and Classification

- Riddhi More, Jeremy S. Bradbury

*[published In the proc. of the International Conference on
Software Testing, Verification and Validation (ICST 2025)]*

What are Flaky Tests?

*Flaky tests exhibit **non-deterministic** behavior during execution, and they may **pass** or **fail** without any changes to the program under test.*

Datasets

- International Dataset of Flaky Tests (IDoFT) [1]
 - Flaky (3195) & non-flaky tests (618)
 - Tests organized by open-source projects
- FlakyCat Dataset [2]
 - Only flaky tests (369)
 - Diverse data from multiple open-source projects

TABLE I: IDoFT and FlakyCat datasets

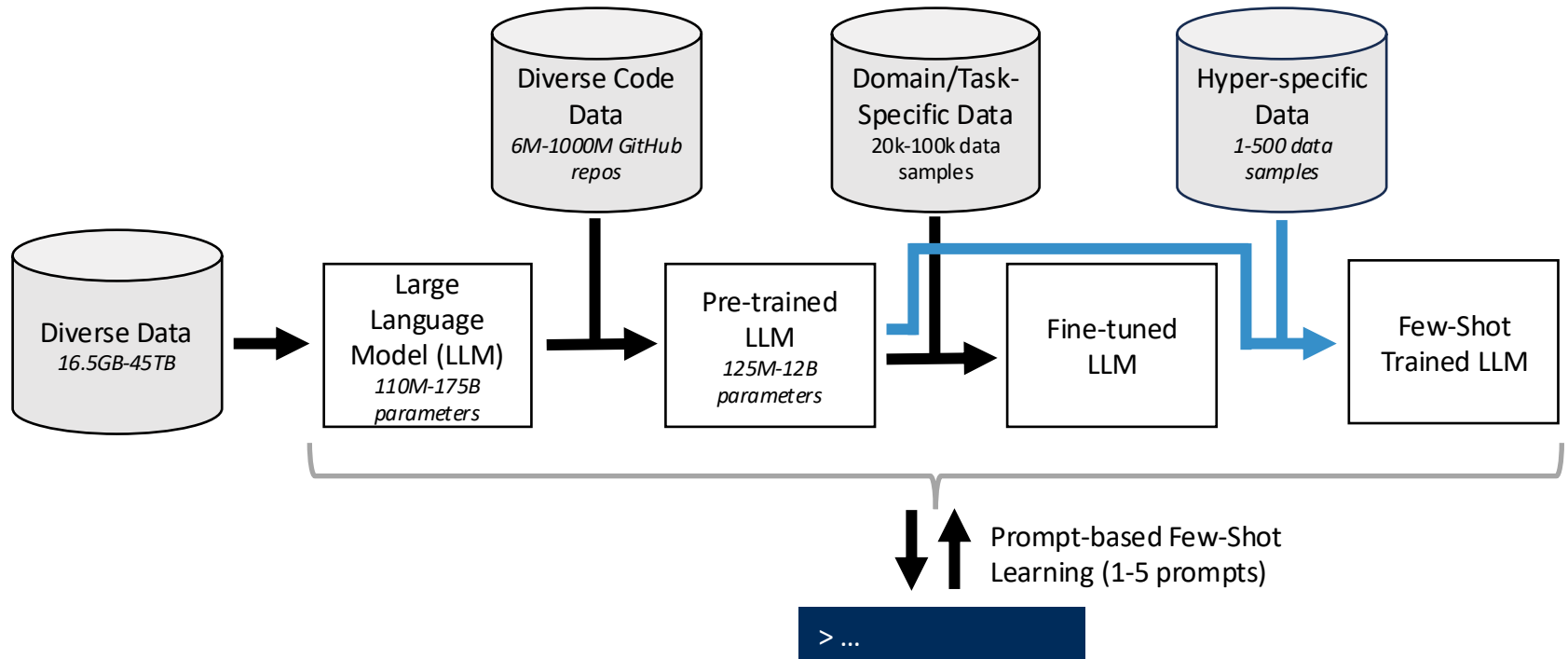
IDoFT - Flaky vs. Non-Flaky	# Tests
Flaky tests	3195
Non-Flaky tests	618
Total	3813
IDoFT - Flaky Test Categories	
Non-deterministic-order-dependent (NDOD)	84
Non-order-dependent (NOD)	226
Order-dependent (OD)	932
Non-idempotent-outcome (NIO)	196
Implementation-dependent (ID)	1617
Unknown-dependency (UD)	140
Total	3195
FlakyCat - Flaky Test Categories	
Async wait (Asyn.)	125
Concurrency (Conc.)	48
Time	42
Test Order Dependency (OD)	103
Unordered Collections (UC)	51
Total	369

[1] “International dataset of flaky tests (IDoFT),” <https://github.com/TestingResearchIllinois/idoft>.

[2] A. Akli, G. Haben, S. Habchi, M. Papadakis, and Y. Le Traon, “Flakycat: predicting flaky tests categories using few-shot learning,” in 2023 IEEE/ACM Int. Conf. on Automation of Software Test (AST 2023), pp. 140–151.

Our research is motivated by the need for an improved understanding of the trade-offs between ***fine-tuning*** and ***few-shot learning (FSL)*** techniques in addressing flaky test challenges.

LLM Training and Usage



Fine-tuning vs FSL

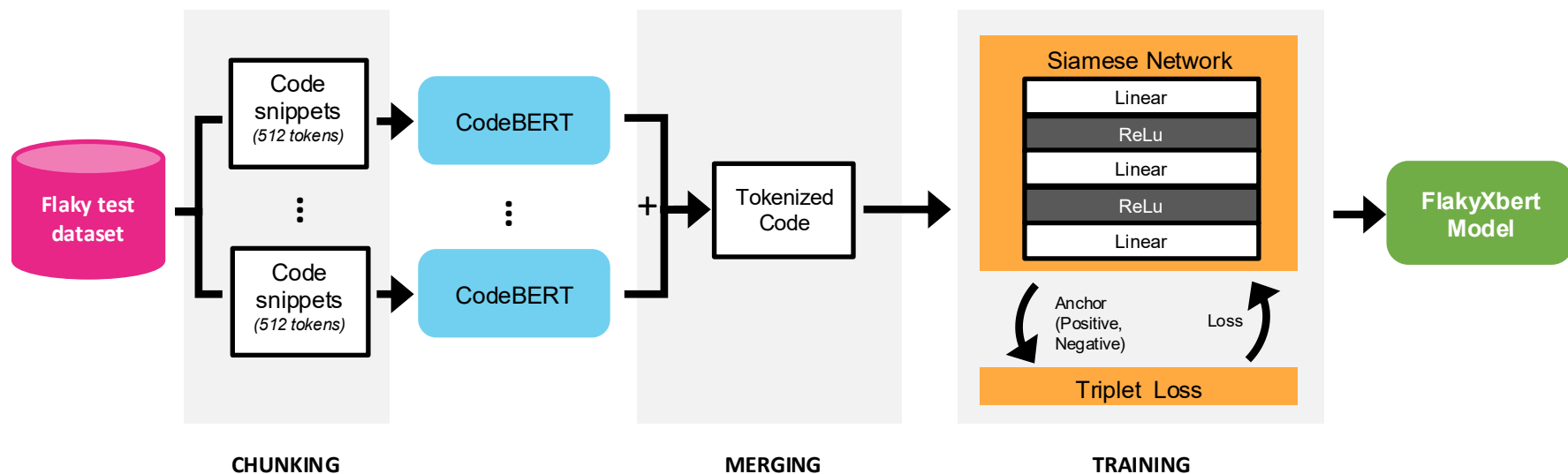
Fine-Tuning:

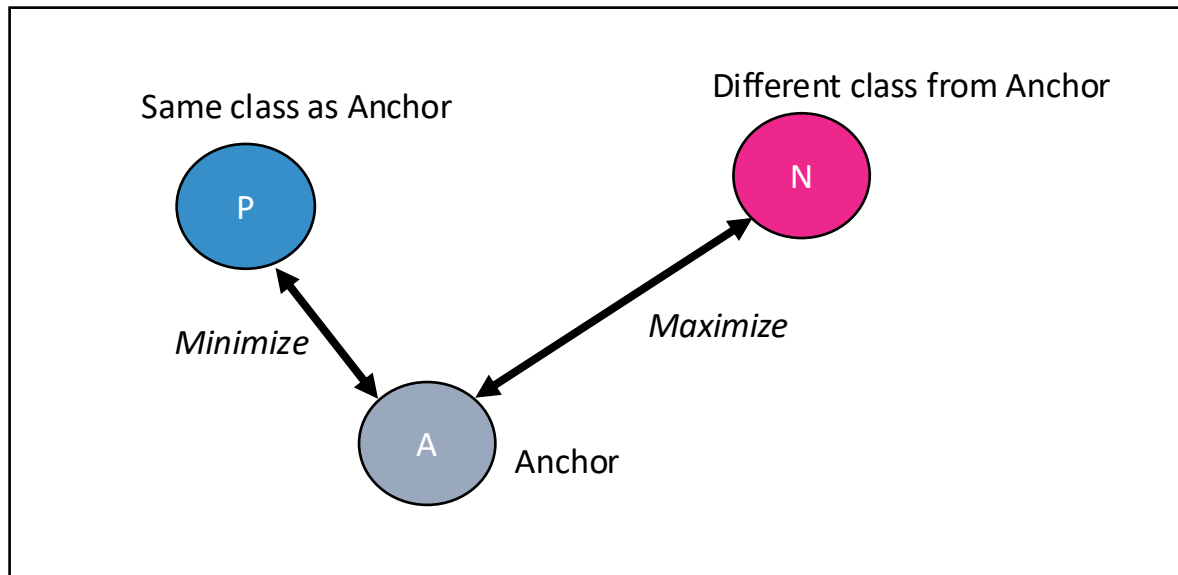
- Adapts pre-trained models to general task-based datasets.
- Requires large labeled datasets for precision.
- Ideal for more global usage
- Enhances model accuracy and robustness
- Very resource-intensive.

Few-Shot Learning (FSL):

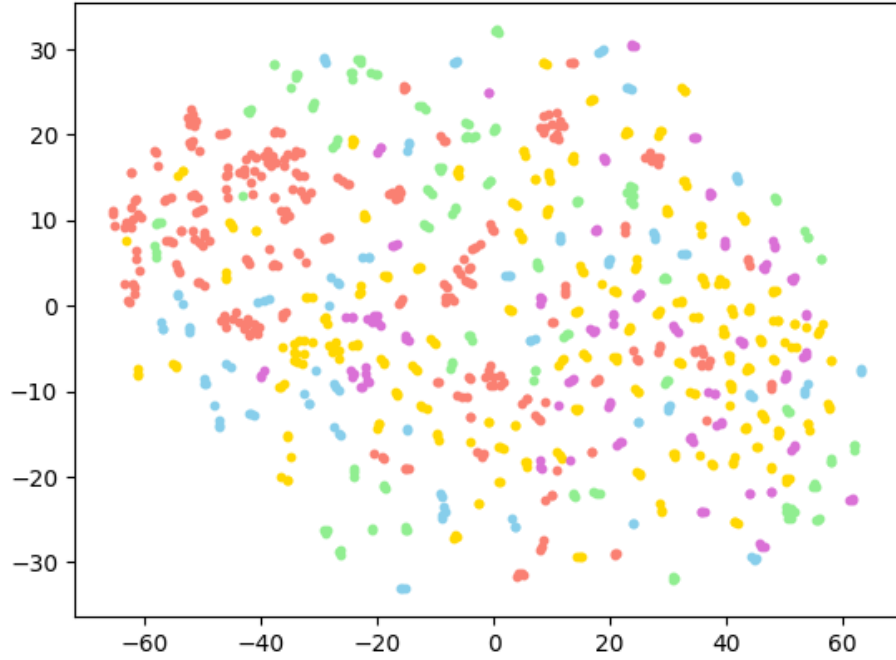
- Adapts model to hyper-specific context
- Works with minimal data.
- Ideal for data-constrained environments or rapid deployment.
- Faster and more flexible
- Does not generalise well.

FlakyXbert: Architecture

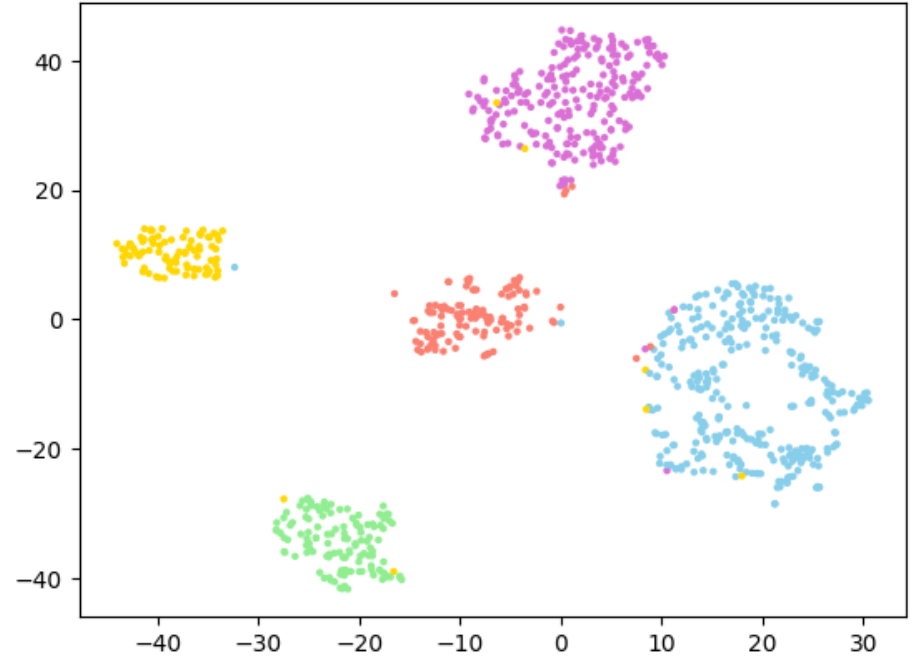




Triplet Loss function



Before FlakyXbert



After FlakyXbert

Research Questions

- **RQ1:** How does the **performance** of FSL and fine-tuning compare for flaky test detection and classification across different data scenarios?
 - **RQ1.1:** What is the performance of FSL compared to fine-tuning on small **per-project data**? (IDoFT)
 - **RQ1.2:** What is the performance of FSL compared to fine-tuning with a **diverse data set**? (FlakyCat)
- **RQ2:** What is the **cost** of FSL vs. fine-tuning?

Results: IDoFT Detection – F1-Score

Project	Support	FlakyXbert	Flakify++	Q-Flakify++	FlakyQ_RF
Dubbo	186	88.7	87.0	91.0	88.0
Hadoop	149	95.0	99.0	100.0	100.0
Nifi	146	91.5	99.0	100.0	100.0
Junit	250	94.0	99.0	99.0	99.0
Admiral	113	91.3	99.0	99.0	99.0
Fastjson	109	91.3	91.0	93.0	93.0
spring	68	100.0	100.0	100.0	100.0
Adyen	89	30.0	43.0	52.0	45.0
Mockserver	39	100.0	100.0	100.0	100.0
Total/ Weighted Avg.	2105	95.1	95.6	96.0	95.4

Note: To see the full version, please refer to the original paper [2].

[2] S. Rahman, A. Baz, S. Misailovic, and A. Shi, “Quantizing large- language models for predicting flaky tests,” in *Proc. of the 17th IEEE Int. Conf. on Software Testing, Verification and Validation (ICST 2024)*.

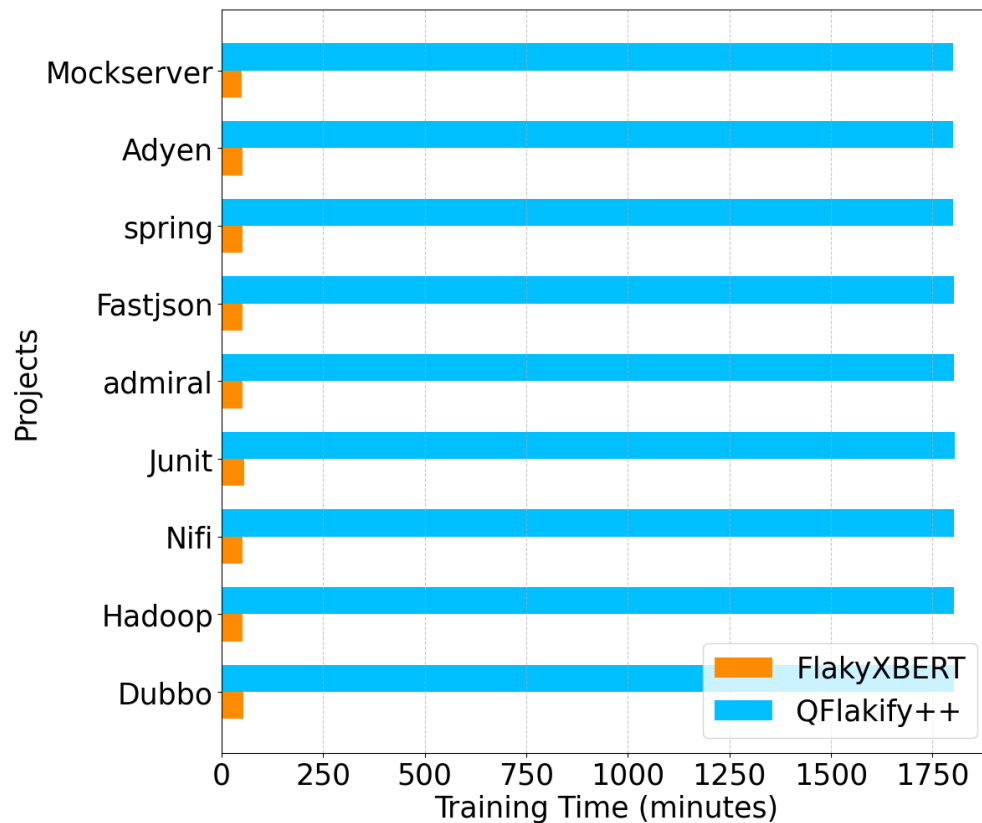
Results: IDoFT Detection – F1-Score

Project	Support	FlakyXbert	Flakify++	Q-Flakify++	FlakyQ_RF
Dubbo	186	88.7	87.0	91.0	88.0
Hadoop	149	95.0	99.0	100.0	100.0
Nifi	146	91.5	99.0	100.0	100.0
Junit	250	94.0	99.0	99.0	99.0
Admiral	113	91.3	99.0	99.0	99.0
Fastjson	109	91.3	91.0	93.0	93.0
spring	68	100.0	100.0	100.0	100.0
Adyen	89	30.0	43.0	52.0	45.0
Mockserver	39	100.0	100.0	100.0	100.0
Total/ Weighted Avg.	2105	95.1	95.6	96.0	95.4

Note: To see the full version, please refer to the original paper [2].

[2] S. Rahman, A. Baz, S. Misailovic, and A. Shi, “Quantizing large- language models for predicting flaky tests,” in *Proc. of the 17th IEEE Int. Conf. on Software Testing, Verification and Validation (ICST 2024)*.

Results: IDoFT Detection – Training Time



Results: IDoFT Classification – F1-Score

Project	Support	FlakyXbert	Flakify++	Q-Flakify++	FlakyQ_RF
Dubbo	170	71.0	77.0	77.0	73.0
Hadoop	146	51.0	90.0	88.0	91.0
Nifi	139	91.0	100.0	100.0	100.0
Junit	250	94.0	98.0	98.0	98.0
Ormlite	113	96.0	99.0	97.0	97.0
admiral	109	63.0	85.0	77.0	88.0
Wildfly	84	74.0	97.0	98.0	98.0
Mapper	75	100.0	93.0	80.0	100.0
Fastjson	64	82.0	91.0	88.0	94.0
Java	54	85.0	87.0	87.0	87.0
Biojava	51	91.0	19.0	16.0	32.0
spring	68	90.0	100.0	100.0	100.0
Hbase	47	76.0	98.0	95.0	98.0
hive	41	100.0	98.0	96.0	98.0
Nacos	32	96.0	100.0	97.0	97.0
Total/ Weighted Avg.	1810	76.5	90.2	93.0	94.8

Note: To see the full version, please refer to the original paper [2]

[2] S. Rahman, A. Baz, S. Misailovic, and A. Shi, “Quantizing large- language models for predicting flaky tests,” in *Proc. of the 17th IEEE Int. Conf. on Software Testing, Verification and Validation (ICST 2024)*.

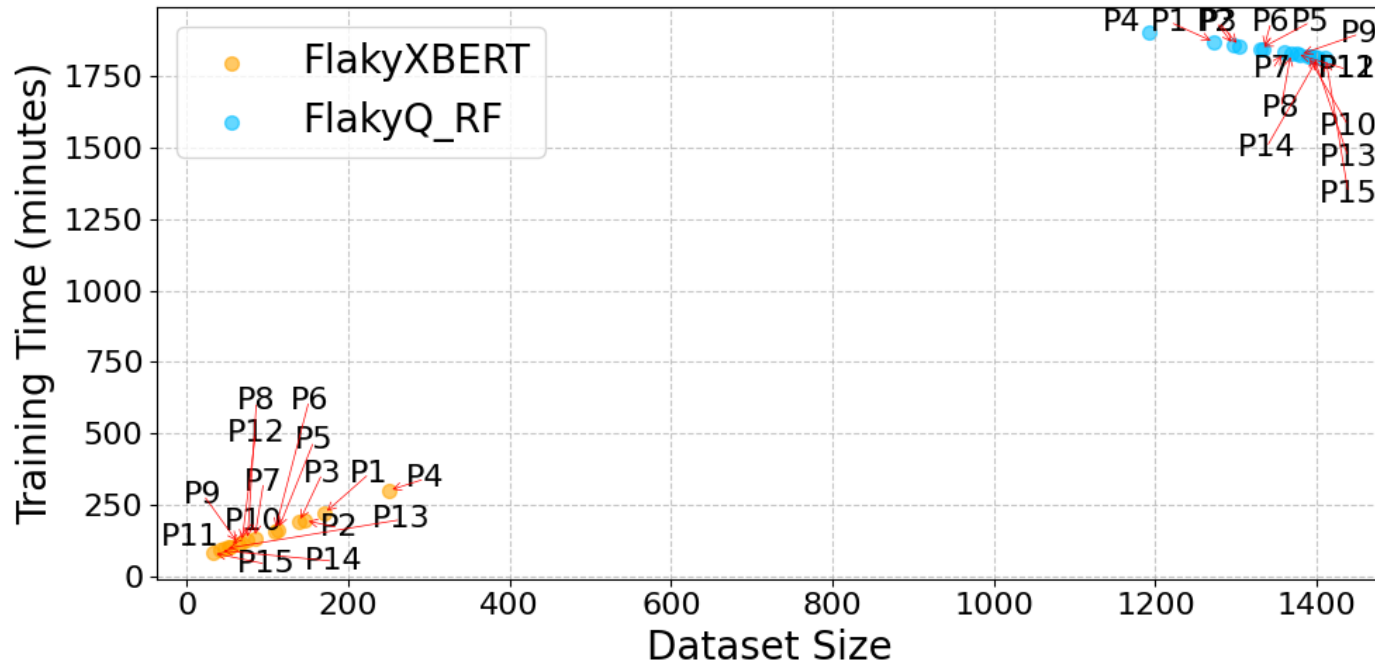
Results: IDoFT Classification – F1-Score

Project	Support	FlakyXbert	Flakify++	Q-Flakify++	FlakyQ_RF
Dubbo	170	71.0	77.0	77.0	73.0
Hadoop	146	51.0	90.0	88.0	91.0
Nifi	139	91.0	100.0	100.0	100.0
Junit	250	94.0	98.0	98.0	98.0
Ormlite	113	96.0	99.0	97.0	97.0
admiral	109	63.0	85.0	77.0	88.0
Wildfly	84	74.0	97.0	98.0	98.0
Mapper	75	100.0	93.0	80.0	100.0
Fastjson	64	82.0	91.0	88.0	94.0
Java	54	85.0	87.0	87.0	87.0
Biojava	51	91.0	19.0	16.0	32.0
spring	68	90.0	100.0	100.0	100.0
Hbase	47	76.0	98.0	95.0	98.0
hive	41	100.0	98.0	96.0	98.0
Nacos	32	96.0	100.0	97.0	97.0
Total/ Weighted Avg.	1810	76.5	90.2	93.0	94.8

Note: To see the full version, please refer to the original paper [2]

[2] S. Rahman, A. Baz, S. Misailovic, and A. Shi, “Quantizing large- language models for predicting flaky tests,” in *Proc. of the 17th IEEE Int. Conf. on Software Testing, Verification and Validation (ICST 2024)*.

Results: IDoFT Classification – Training Time vs. Dataset Size



Results: FlakyCat Classification – F1-Score

Technique	Classifier	Asyn.	Conc.	Time	UC	OD	Weighted Avg.
Few-shot Learning (FSL)	FlakyXbert	98.0	90.0	93.0	97.0	99.0	96.0
	FlakyXbert (without augmentation)	52.0	80.0	36.0	43.0	78.0	60.0
	FlakyCat	72.0	36.0	75.0	72.0	73.0	67.5
Fine-tuning (FT)	Flakify++	94.8	93.3	96.9	96.1	97.1	95.6
	Q-Flakify++	92.6	87.1	96.9	95.0	95.8	93.6
	FlakyQ_KNN	93.1	90.7	95.5	95.0	96.3	94.2
	FlakyQ_MLP	94.0	89.7	95.5	94.8	96.6	94.5
	FlakyQ_RF	94.3	91.5	95.5	94.8	96.6	94.8
	FlakyQ_SVM	93.8	89.0	95.5	93.2	96.1	93.9
	FlakyQ_LR	92.7	89.8	95.5	94.8	96.1	93.9
Hybrid (FSL + FT)	FSL++	93.7	90.3	97.9	95.9	96.7	91.5

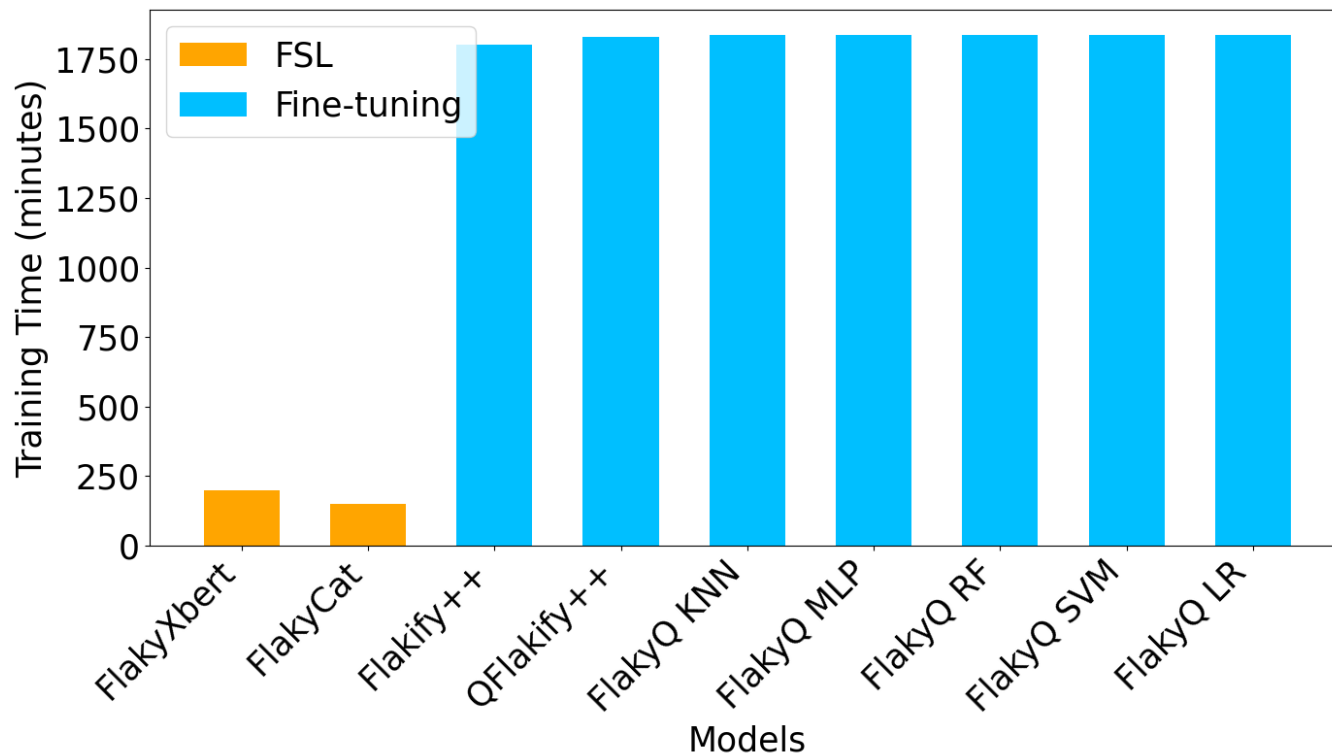
Note: Asyn. = Async Wait, Conc. = Concurrency, Time = Test Order Dependency, UC = Unordered Collections, OD = Other Dependencies

Results: FlakyCat Classification – F1-Score

Technique	Classifier	Asyn.	Conc.	Time	UC	OD	Weighted Avg.
Few-shot Learning (FSL)	FlakyXbert	98.0	90.0	93.0	97.0	99.0	96.0
	FlakyXbert (without augmentation)	52.0	80.0	36.0	43.0	78.0	60.0
	FlakyCat	72.0	36.0	75.0	72.0	73.0	67.5
Fine-tuning (FT)	Flakify++	94.8	93.3	96.9	96.1	97.1	95.6
	Q-Flakify++	92.6	87.1	96.9	95.0	95.8	93.6
	FlakyQ_KNN	93.1	90.7	95.5	95.0	96.3	94.2
	FlakyQ_MLP	94.0	89.7	95.5	94.8	96.6	94.5
	FlakyQ_RF	94.3	91.5	95.5	94.8	96.6	94.8
	FlakyQ_SVM	93.8	89.0	95.5	93.2	96.1	93.9
	FlakyQ_LR	92.7	89.8	95.5	94.8	96.1	93.9
Hybrid (FSL + FT)	FSL++	93.7	90.3	97.9	95.9	96.7	91.5

Note: Asyn. = Async Wait, Conc. = Concurrency, Time = Test Order Dependency, UC = Unordered Collections, OD = Other Dependencies

Results: FlakyCat Classification – Training Time



Conclusion

FSL (Few-Shot Learning) in the FlakyXbert model is effective in environments with sparse data.

- It leverages fewer labelled examples to classify and predict flakiness categories
- **Small scale companies and research labs can benefit**
- Retraining is required for a different project or use case

Conclusion

Fine-tuning, which requires more extensive data, excels by adapting to a broader range of features.

- It handles diversity better and typically offers more accurate predictions
- It demands greater computational resources and longer training time
- Minimal/no retraining is required when it comes to another project or use case

Conclusion

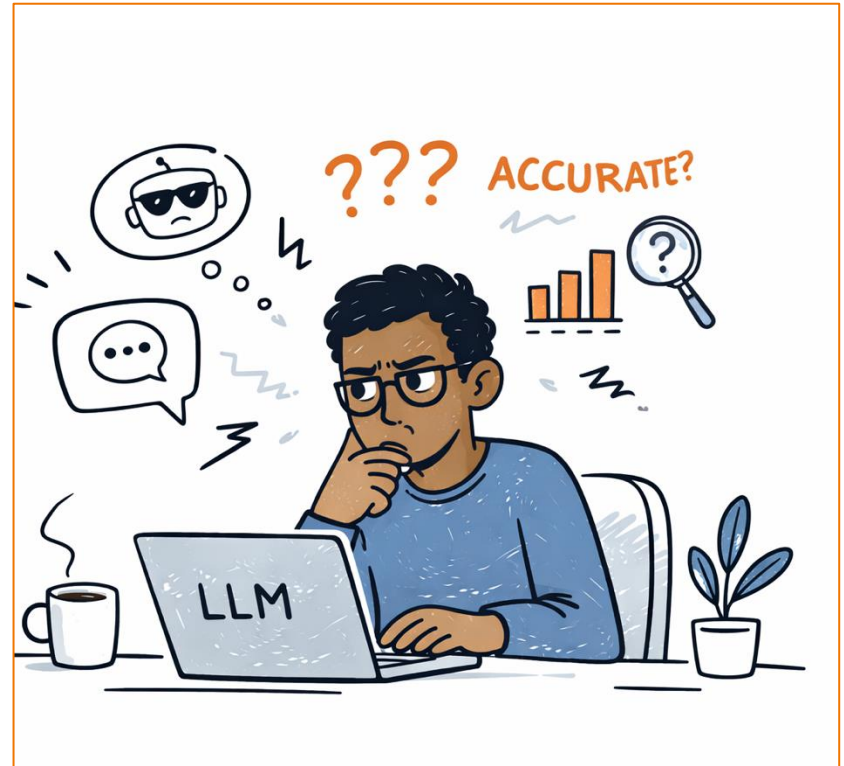
- The choice between ***FSL*** and ***Fine-tuning*** depends on balancing trade-offs between:
 - Data availability.
 - Computational efficiency.
 - Accuracy of Results
 - Adaptability to diverse flaky test characteristics.

Threats to Validity & Future Work

- We used publicly available datasets and consistent evaluation metrics across models
- Addressed class imbalances with augmentation – can lead to bias
- While further tuning could improve performance, variability in results across projects and uncertainty in generalizing our findings remain.
- More experimentation is needed to confirm broader applicability beyond flaky test detection.

Lack of Trust in Results

- **Trust** → Is there confidence in the quality and correctness of results?
- **Benchmarks** → Are there sufficient benchmarks available to demonstrate effectiveness of LLM on a given task?
- **Data Leakage** → Has the benchmark been compromised?



Addressing Data Leakage in HumanEval using Combinatorial Test Design

- Jeremy S. Bradbury, Riddhi More

*[published In the proc. of the International Conference on
Software Testing, Verification and Validation (ICST 2025) – Short
Papers, Vision and Emerging Results Track]*

Benchmarking & LLMs

- **Benchmarks** are standardized evaluation tools that enable systematic comparison of different approaches that solve the same problem or task [1]
- LLM benchmarks that have experienced **data leakage** (when benchmark data leaks into the training data) will likely exhibit inflated benchmark evaluation scores which can misrepresent their ability to address the underlying task [2]

[1] S. Sim, S. Easterbrook, and R. Holt, “Using benchmarking to advance research: a challenge to software engineering,” in Proc. of the 25th International Conference on Software Engineering (ICSE 2003), May 2003, pp. 74–83.

[2] O. Sainz, J. Campos, I. Garc´ıa-Ferrero, J. Etxaniz, O. L. de Lacalle, and E. Agirre, “NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark,” in Findings of the Association for Computational Linguistics (EMNLP), Dec. 2023, pp. 10 776–10 787.

Benchmarking & LLMs: Best Practices

- Benchmark **Construction**
 - Tasks can not originate from sources that are part of LLM training data
 - (1) hand-craft tasks (e.g., HumanEval)
 - (2) tasks selected from private data sets
- Benchmark **Operation**
 - All benchmark data must be actively excluded from future LLM training data sets.

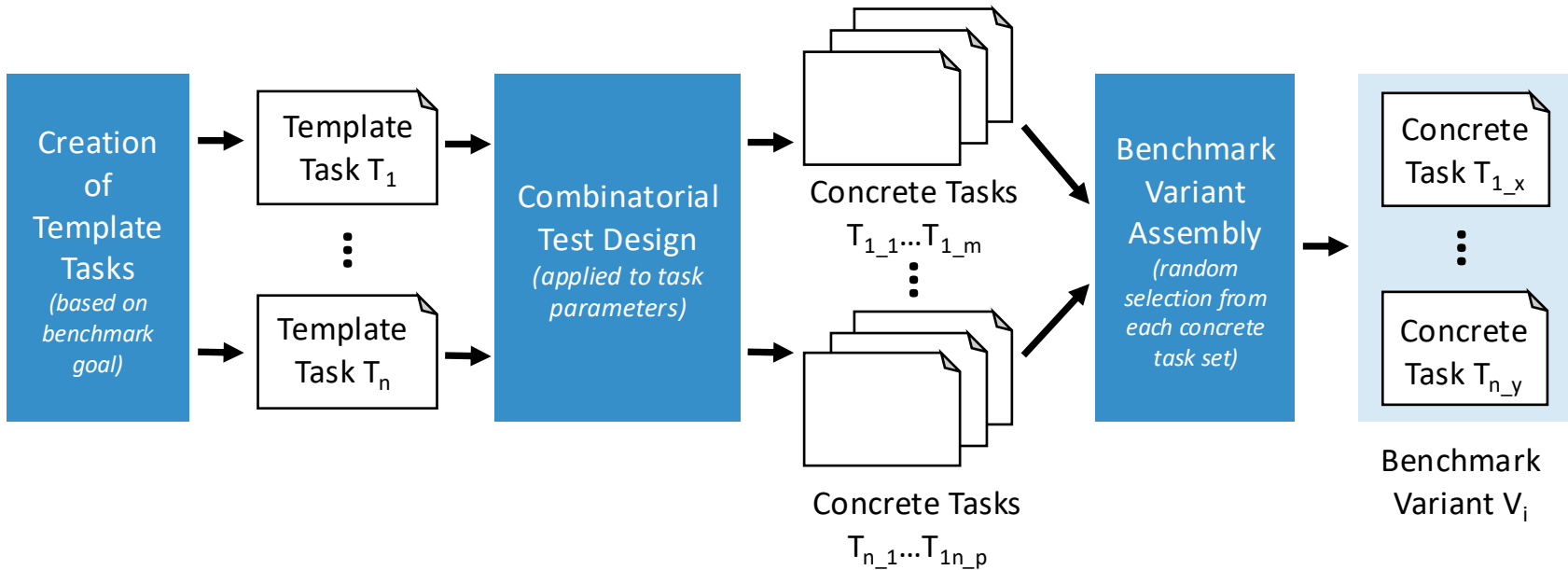
HumanEval Benchmark

- Developed in 2021 at OpenAI [3]
- Assesses LLMs with respect to **program generation**
- Contains 164 hand crafted tasks
- Example:

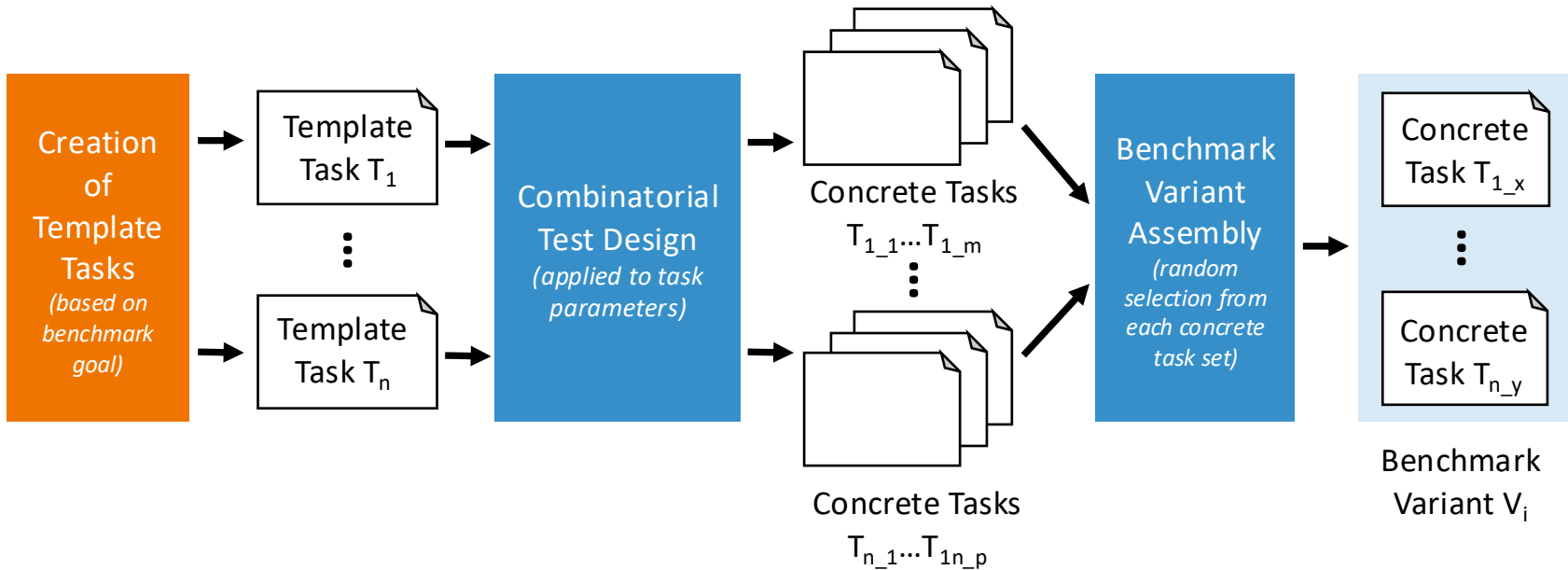
```
Given a list of numbers and a
threshold value, determine if any two
values are closer than the threshold.
...
```

[3] M. Chen, et al. "Evaluating large language models trained on code," 2021. [Online].
Available: <https://arxiv.org/abs/2107.03374>

Our Benchmark Construction Process



Our Benchmark Construction Process



Creation of Template Tasks

Given a list of numbers and a threshold value, determine if any two values are closer than the threshold.

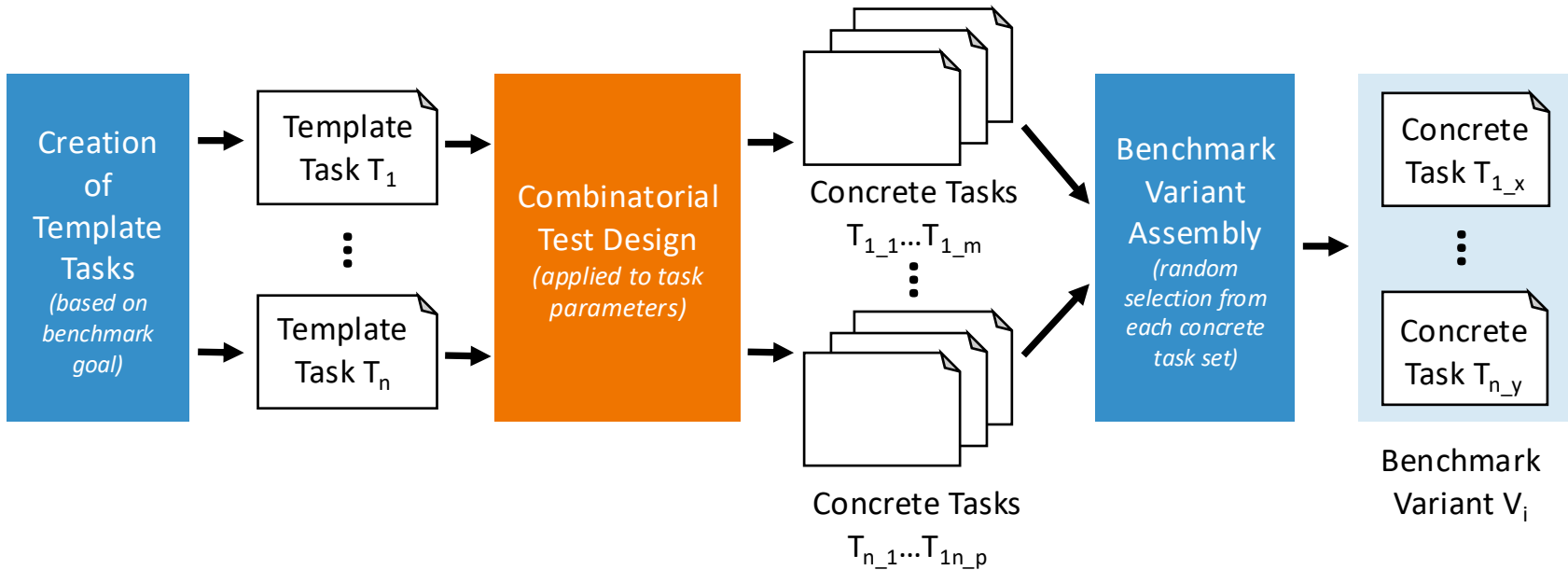


Given a list of **<input_type>** and **<threshold_descriptor>**, check if any two **<value_descriptor>** are closer than the given **<threshold_descriptor>**.

Where:

- **<input_type>**: numbers, float values, measurements
- **<threshold_descriptor>**: threshold, minimum distance, tolerance
- **<value_descriptor>**: values, elements, data points

Our Benchmark Construction Process



Creation of Template Tasks

Given a list of `<input_type>` and `<threshold_descriptor>`,
check if any two `<value_descriptor>` are closer than
the given `<threshold_descriptor>`.

...

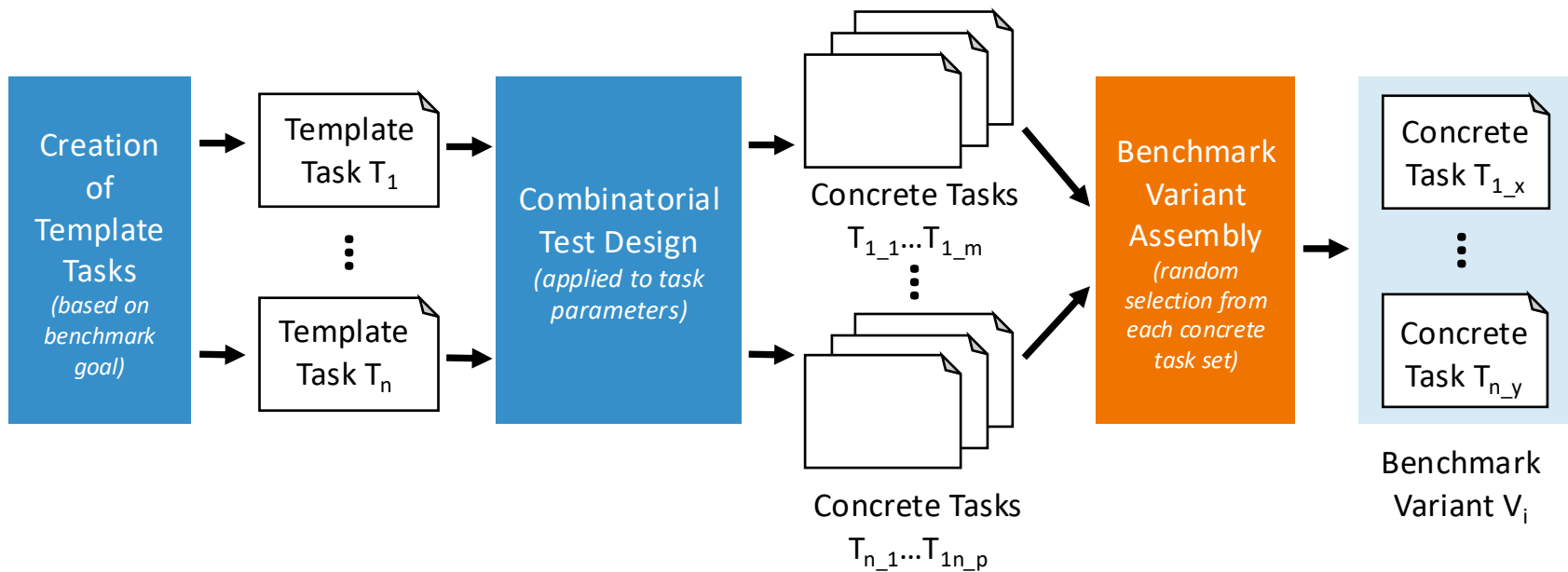


Given a list of **numbers** and a **threshold**, check if any two
values are closer than the given **threshold**

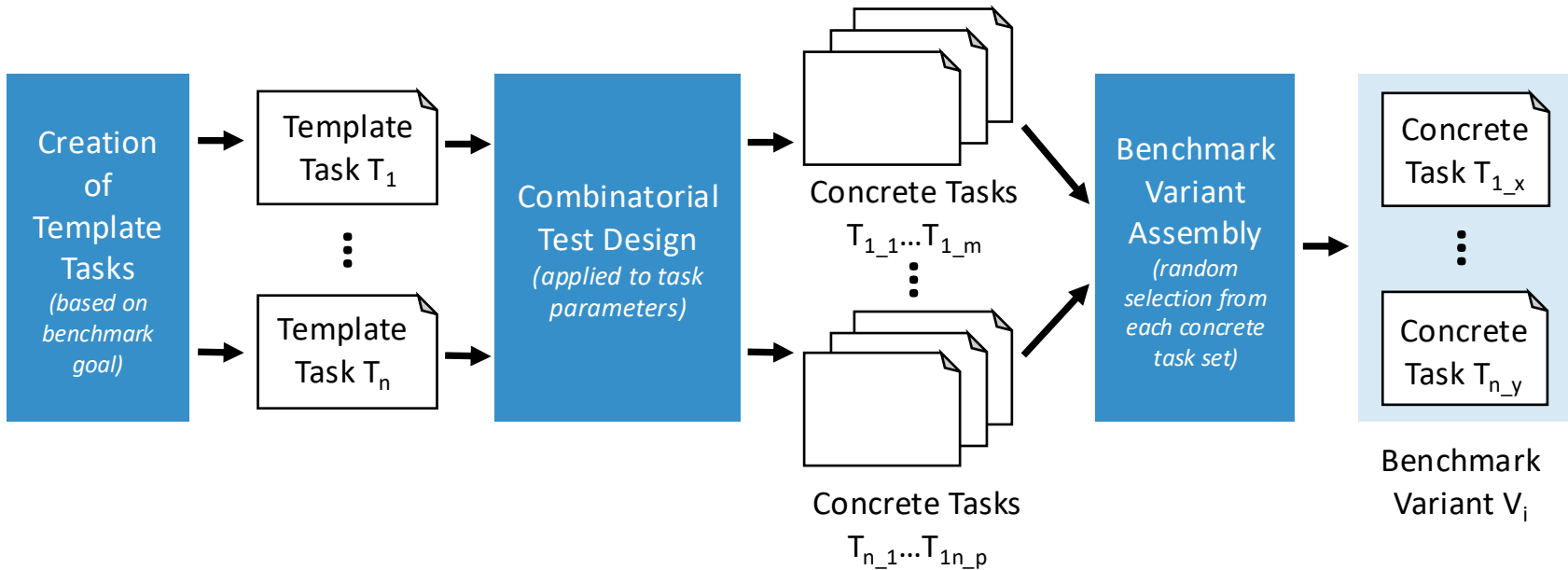
...

Given a list of **measurements** and a **minimum distance**, check
if any two **data points** are closer than given **minimum
distance**

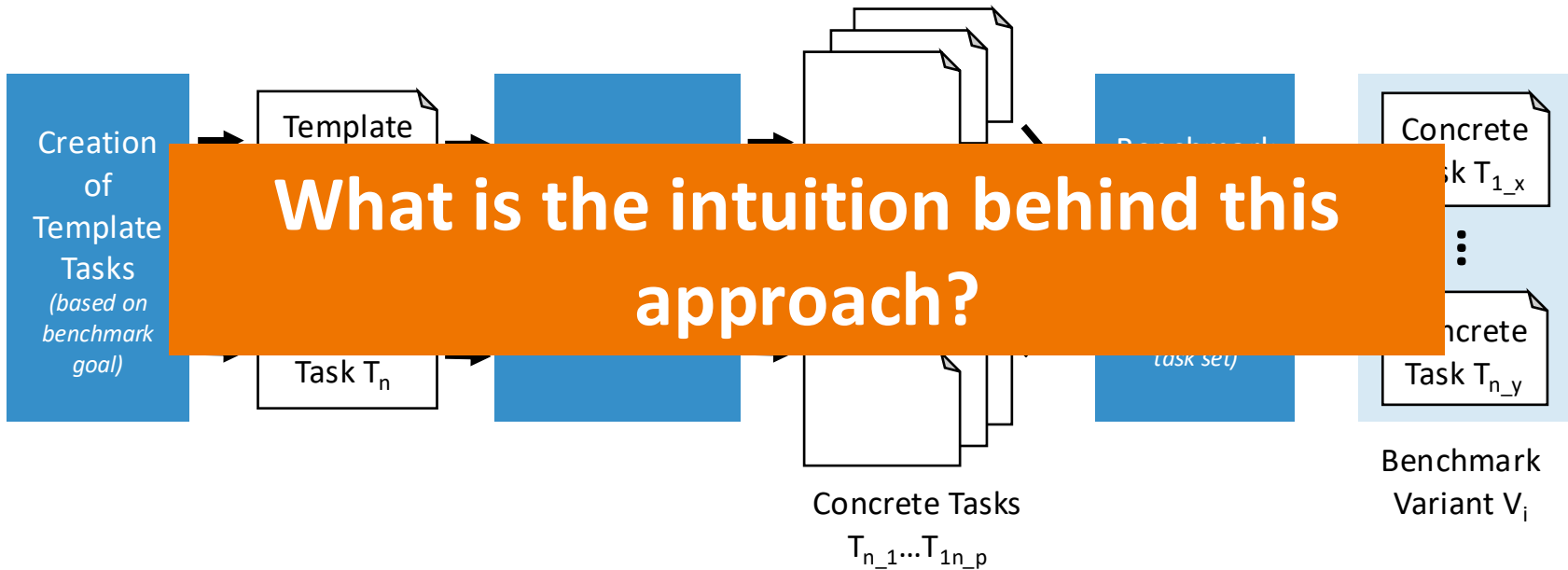
Our Benchmark Construction Process



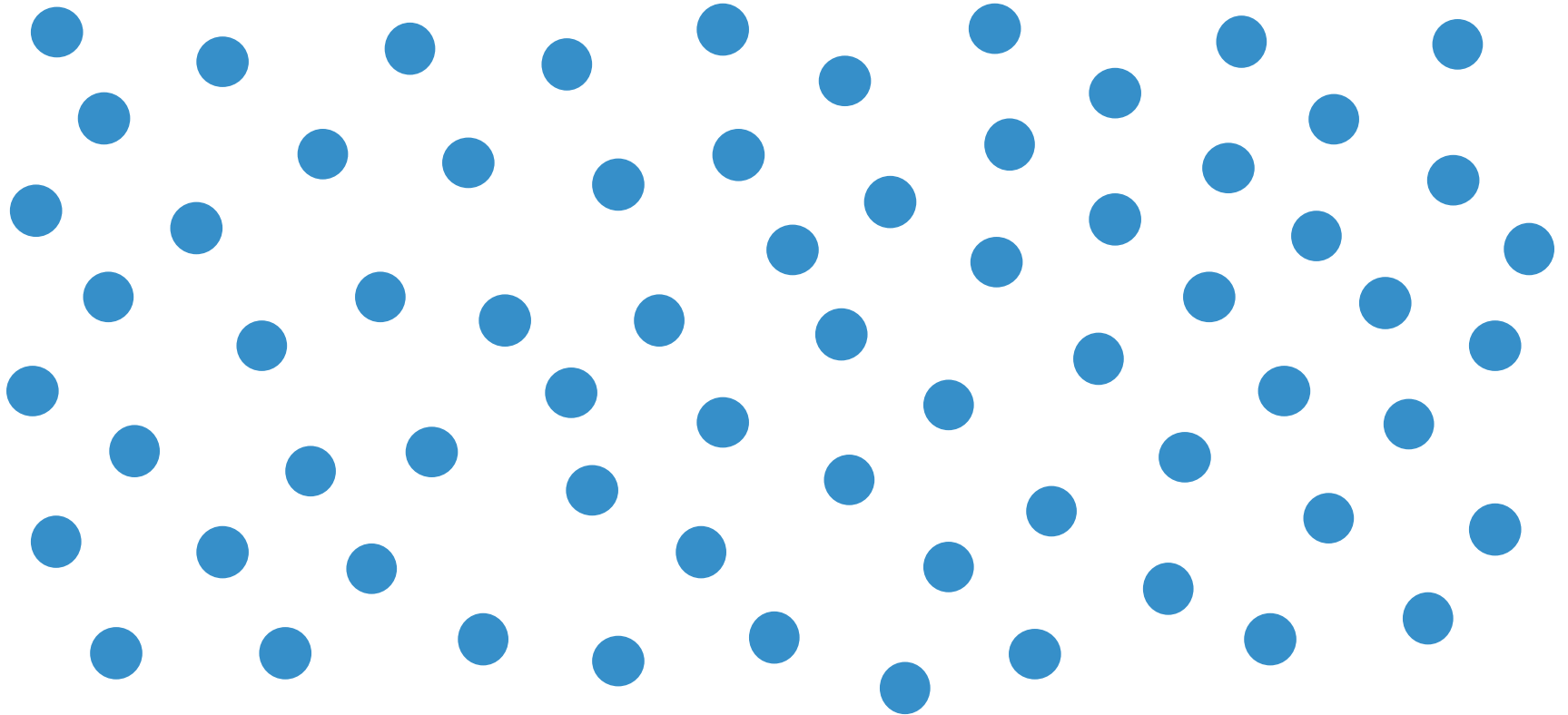
Our Benchmark Construction Process



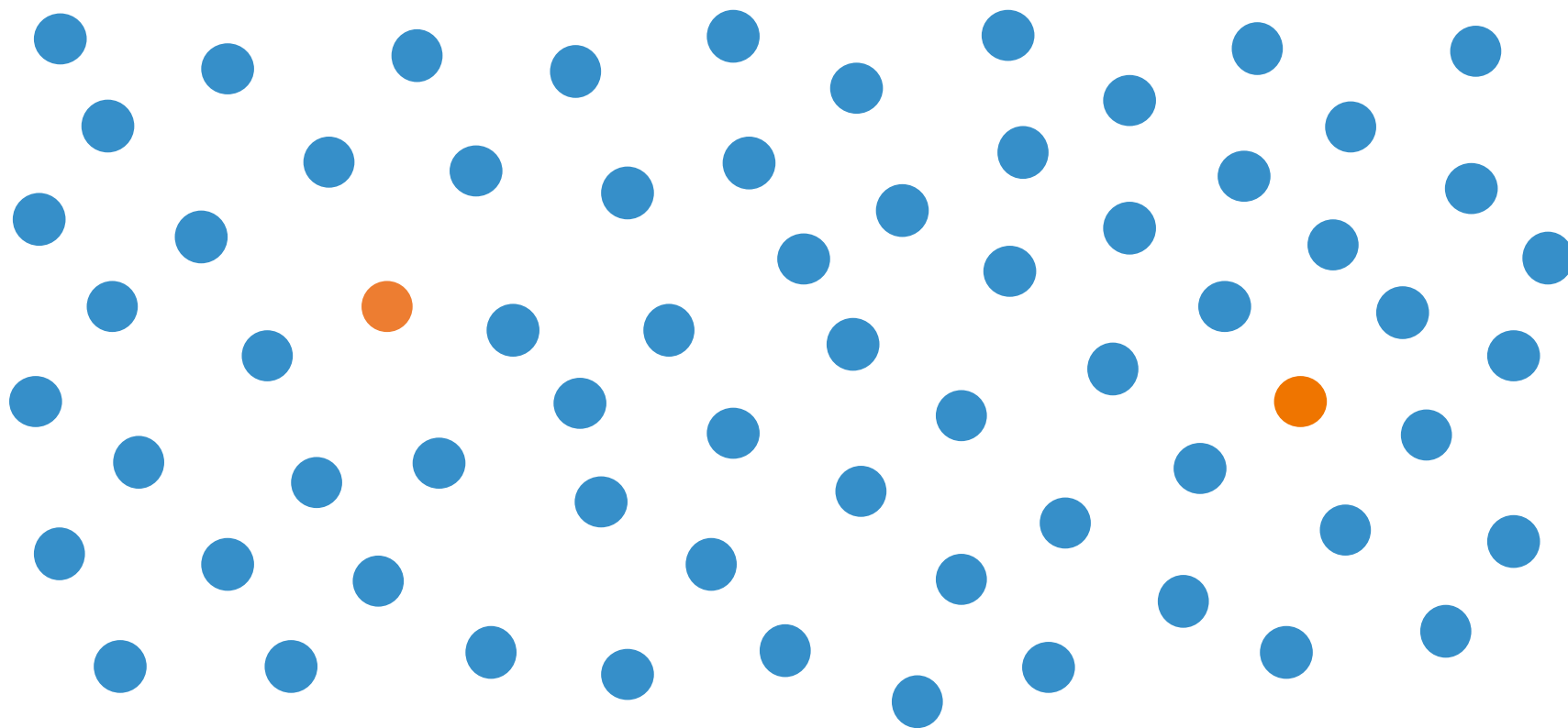
Our Benchmark Construction Process



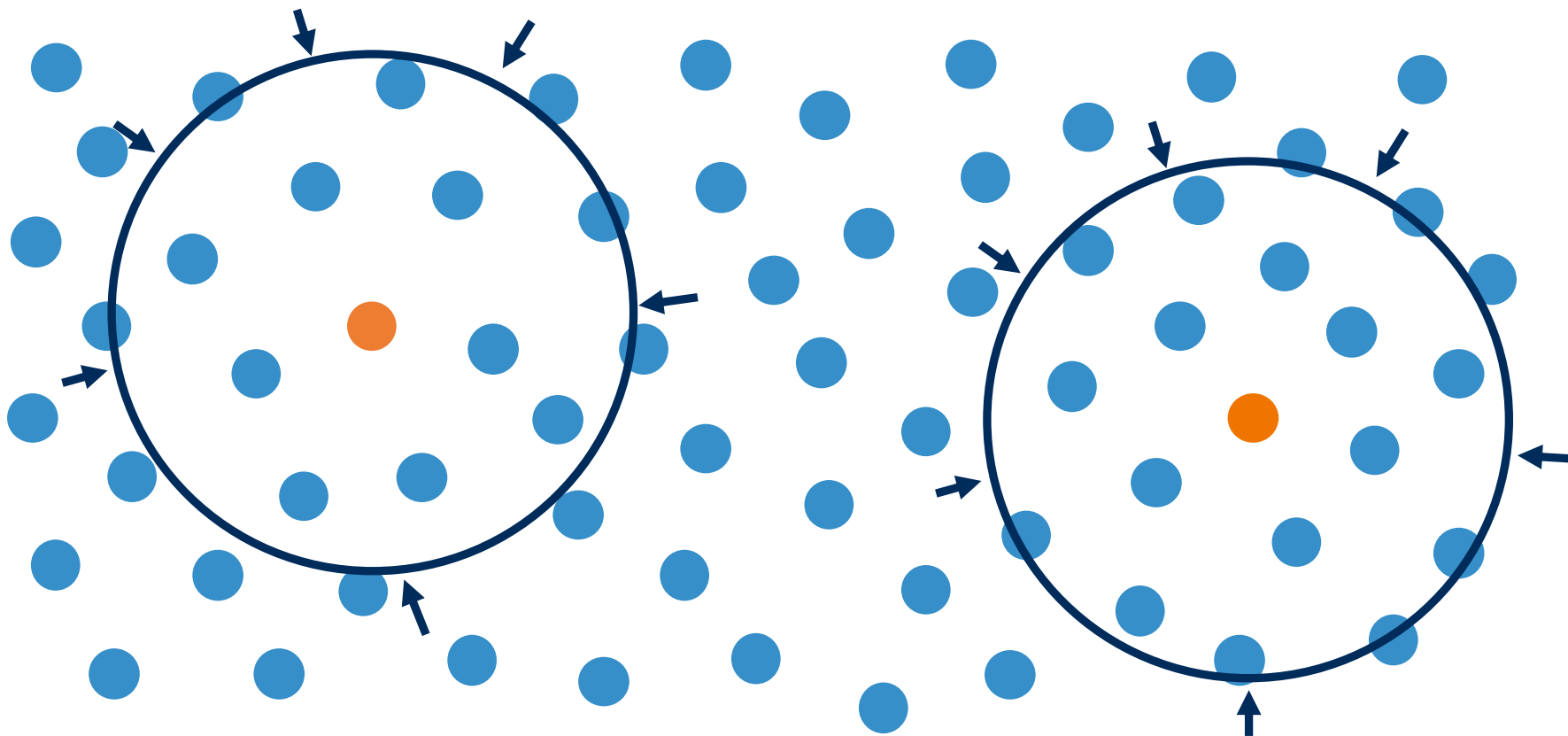
Program Generation Problem Space



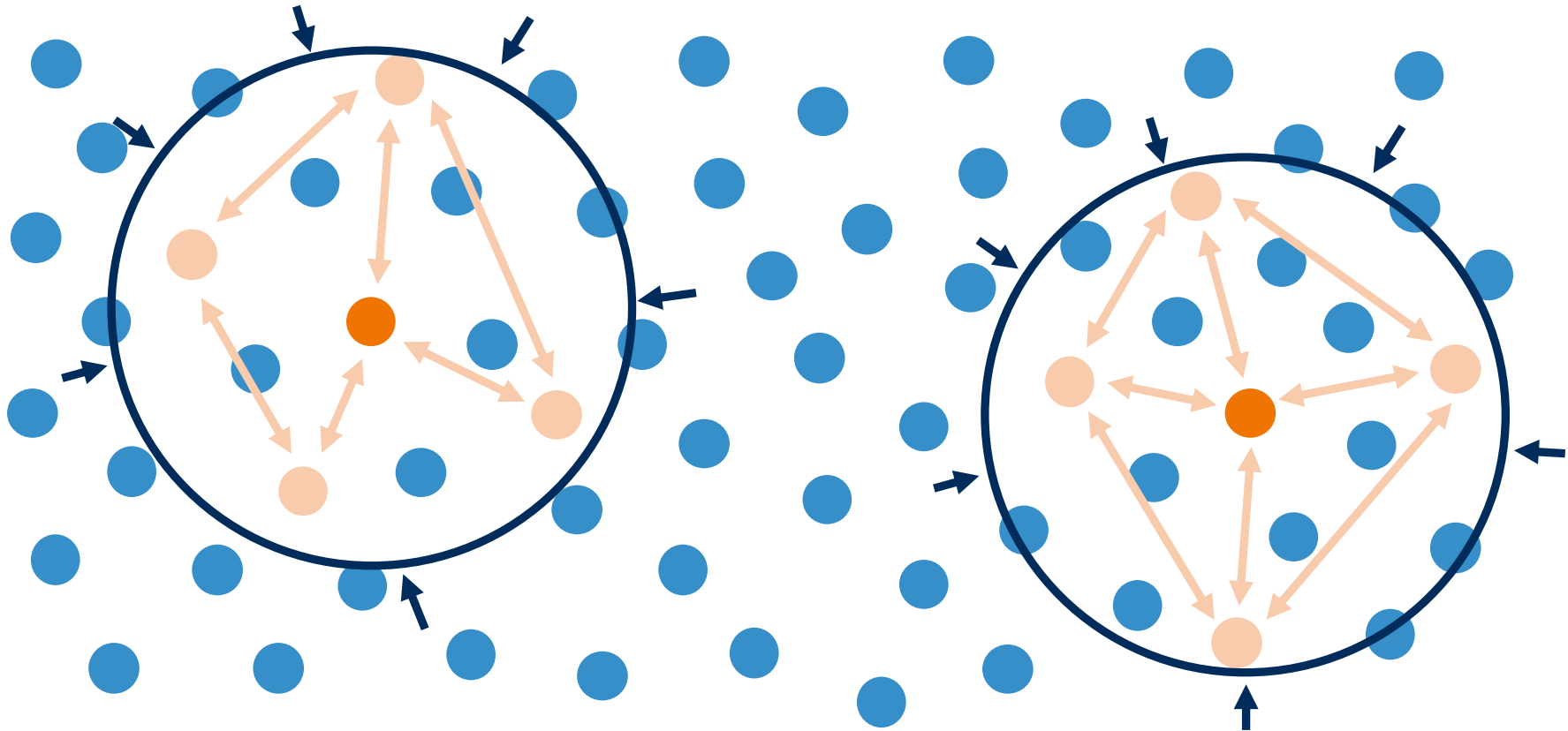
HumanEval Program Generation Tasks



HumanEval_T Template Tasks



HumanEval_T Concrete Tasks



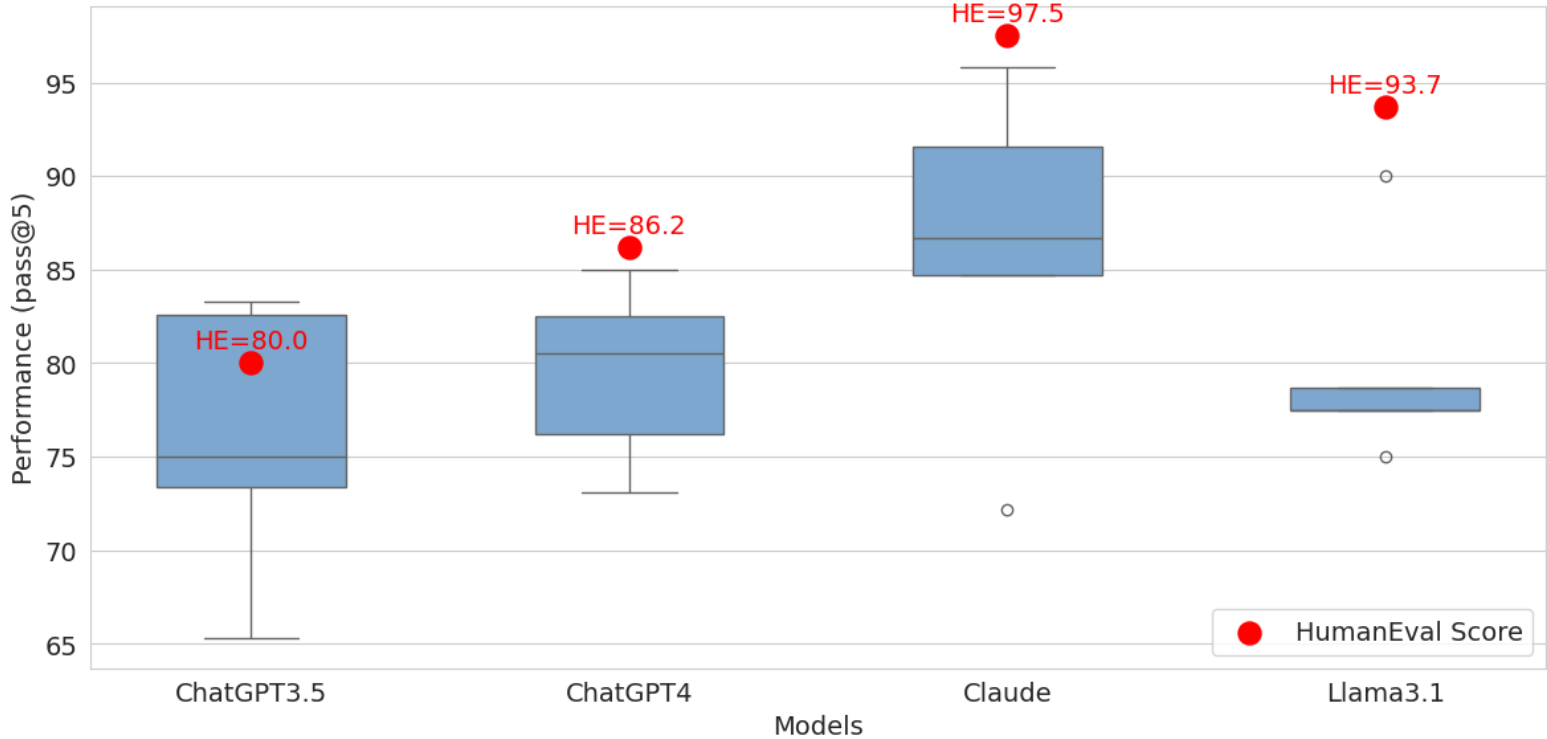
Experimental Design

- Randomly selected 10 **HumanEval** tasks as a baseline
- Created corresponding template tasks by hand
- Generated five unique **HumanEval_T** benchmark variants
- Evaluated the HumanEval tasks and the HumanEval_T variants on four LLMs: **GPT3**, **GPT4**, **Claude**, **Llama3.1**

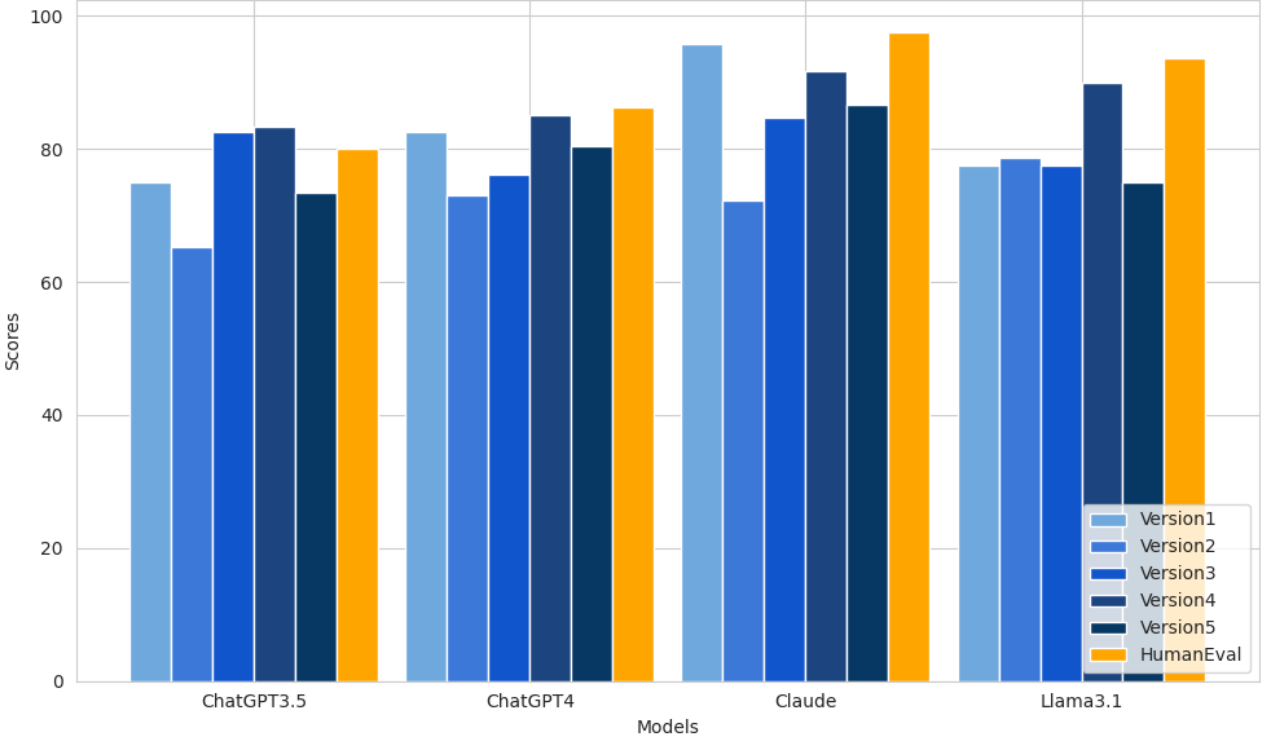
RQ1: Is there evidence of data leakage in HumanEval?

RQ2: Do concrete versions of the same template task in HumanEval_T produce similar results?

RQ1: Is there evidence of data leakage in HumanEval?



RQ2: Do concrete versions of the same template task in HumanEval_T produce similar results?



Conclusions

- We observed ***evidence of data leakage*** in the **HumanEval** benchmark
- Our benchmark construction process systematically creates **HumanEval_T** benchmark variants that are ***robust to data leakage*** from HumanEval
- Low variance in **HumanEval_T** benchmark variant performance is a positive indicator that they ***are similar in difficulty***

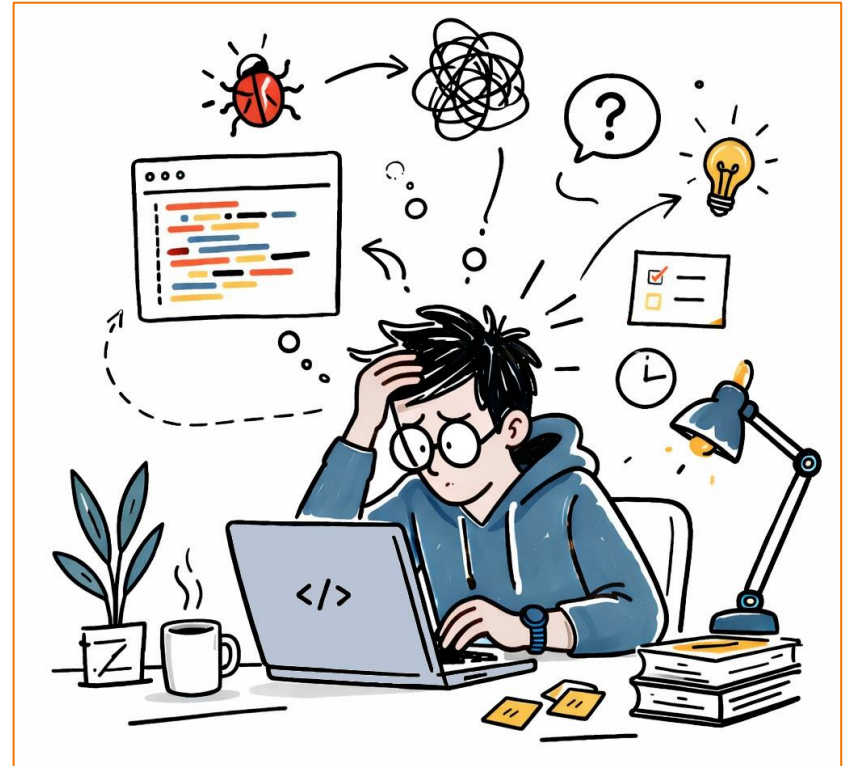
Conclusions

- We observed ***evidence of data leakage*** in the **HumanEval** benchmark
- Our benchmark construction process systematically creates **HumanEval_T** benchmark variants that are ***robust to data leakage*** from HumanEval
- Low variance in **HumanEval_T** benchmark variant performance is a positive indicator that they ***are similar in difficulty***

Open Question: How do we evaluate concrete tasks from the same template tasks to assess their suitability as equivalent tasks for assessment?

Lack of Expertise in Technology

- **Programming** → Is there sufficient expertise in the organization to write programs that leverage LLM APIs?
- **Training** → Is there sufficient expertise to pretrain and fine-tune LLMs?



LLBlocks: A Block-Based Language for LLM Programming

- Bridget Green, Jeremy S. Bradbury
[in progress]

Scaffolding Programming with LLMs

- A growing number of companies are integrating AI in their workplace [1], affecting both how we write computer programs and the types of programs that are written.
- This proliferation of **LLMs in programming** has required new educational curriculum, materials and tools to support learning.
- Programming is **well-scaffolded** with methods including Parsons problems [2] and block-based programming (e.g., Scratch [3]) to support learning computational thinking and fundamentals.

[1] Mohaimenul A., et al. 2024. A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges. *IEEE Access* 12 (2024), 26839–26874.

[2] Ericson, B. J., et al. 2022. Parsons problems and beyond: Systematic literature review and empirical study designs. *Proc. of the 2022 working group reports on innovation and technology in Computer Science education*, 191–234.

[3] John Maloney, et al. 2010. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* 10, 4, Article 16 (Nov. 2010), 15 pages.

LLBlocks Design

- Designed by analyzing student written programs
- Intended to assist learners write programs that leverage **LLM APIs** and **prompting**
- Allows a learner to practice data input/output, prompt construction, prompt patterns and LLM common metrics/strategies (e.g., pass@k)

Block Types	Input Blocks: Blocks that deal with defining and processing data used as input to an LLM.
	Output Blocks: Blocks that define the storage and analysis of data output from an LLM.
	LLM Programming Blocks: Blocks that deal with the core functionality of LLM-enhanced programming, including the definition of LLM prompts.
	General Purpose Blocks: These blocks are consistent with blocks available in general-purpose block-based programming and support constructs such as branching and looping.

```

import os
from dotenv import load_dotenv
import openai
import pandas as pd
import chardet

load_dotenv()
openai.api_key = os.getenv("API_KEY")

# Define the file paths
input_file = "IDoFT_Mapper.csv" # Input CSV file
output_file = "zero_shot_results.csv" # Output file for results

with open("IDoFT_Mapper.csv", "rb") as f:
    result = chardet.detect(f.read())
    print(result)

# Zero-shot learning function
def zero_shot_learning_experiment(input_file, output_file, pass_number):
    try:
        data = pd.read_csv(input_file, encoding='MacRoman')
        required_columns = {'full_code', 'flaky'}
        prompt = 'Classify the test as Flaky(1) or Not Flaky(0). Respond with only a 1 or 0. Here is the test:'
        if not required_columns.issubset(data.columns):
            raise ValueError(f"The CSV file must contain these columns: {required_columns}")
        # Prepare to store results
        results_df = data.copy()

        # Iterate over each test case
        for pass_number in range(1, pass_number+1):
            print(f'Pass Number: {pass_number}')
            # Stores result for each pass
            prompted_outcomes = []

            for index, row in data.iterrows():
                test_case = row['full_code']

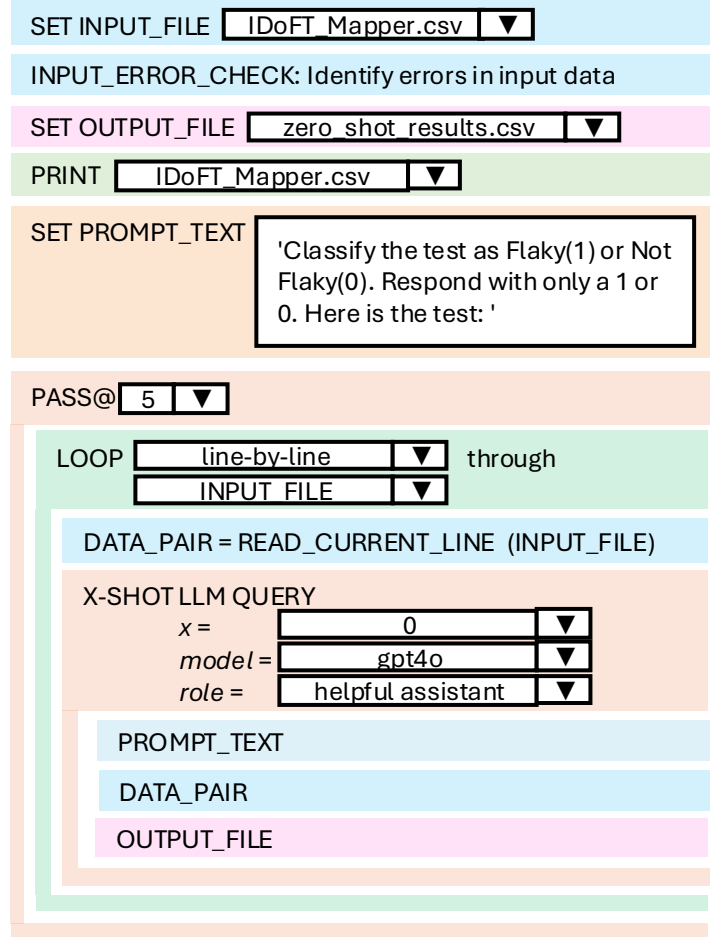
                # Query the model with the prompt
                response = openai.ChatCompletion.create(
                    model="gpt-4o-mini",
                    messages=[
                        {"role": "system", "content": "You are a helpful assistant."},
                        {"role": "user", "content": prompt + test_case}
                    ]
                )
                prompted_outcome = response.choices[0].message["content"]
                prompted_outcomes.append(prompted_outcome)

            # print(f'Processed test case {index + 1}/{len(data)}: {test_case}')
            results_df[f'Pass_{pass_number}'] = prompted_outcomes

        results_df.to_csv(output_file, index=False, encoding='utf-8', errors='ignore')
        print(f'Results saved to {output_file}')
        except Exception as e:
            print(f'An error occurred: {e}')

# Run the experiment
zero_shot_learning_experiment(input_file, output_file, 5)

```



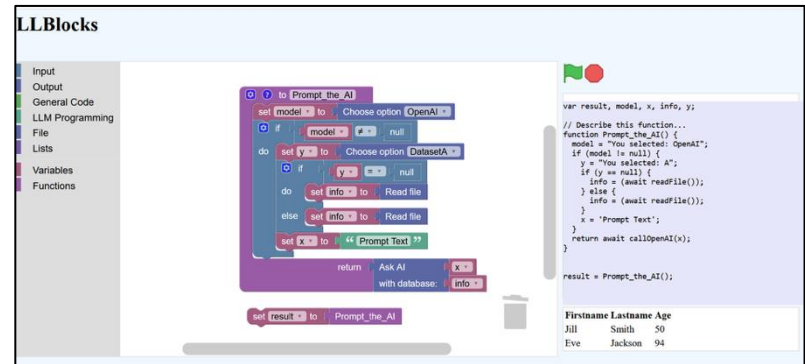
LLBlocks: Current Status & Next Steps

Current Status

- LLBlocks language has been designed to include **common prompting patterns**.
- All LLBlocks blocks have a corresponding Python representation which implicitly defines their **semantics**

Next Steps

- A **web interface** for block-based programming (Blockly API)
- a **user study** to evaluate the **learning** benefits



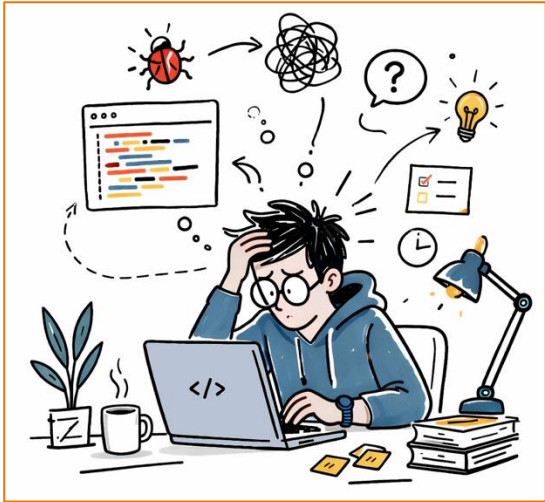
What Limits Democratization of AI in Software Development?



Lack of Resources
(data, computing)



Lack of Trust in
Results

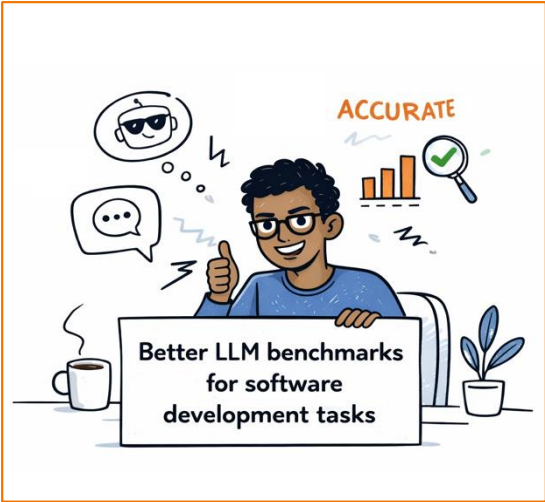


Lack of Expertise
in Technology

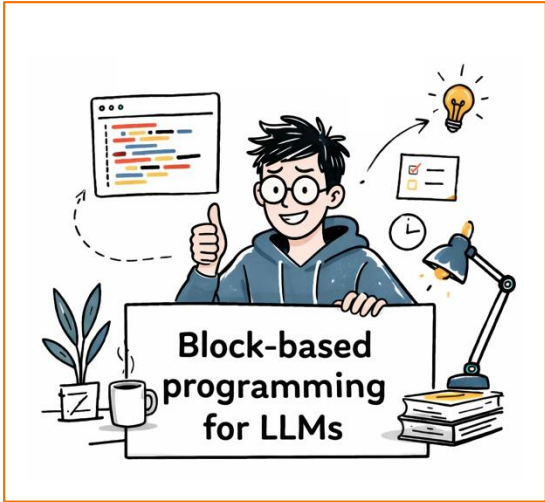
What Limits Democratization of AI in Software Development?



**Lack of Resources
(data, computing)**



**Lack of Trust in
Results**



**Lack of Expertise
in Technology**



Democratizing AI in Software Development

From Few-Shot Testing to Trustworthy Benchmarks and Accessible AI Tools

Jeremy S. Bradbury

Software Engineering & Education Research Lab
Ontario Tech University, Oshawa, Canada

<http://www.seerlab.ca>

SEER*LAB

mairi

OntarioTech
UNIVERSITY

